REDUCED POWER OPTION

BACKGROUND OF THE INVENTION

Field of the Invention:

The present invention relates to processors and methods for instruction processing and, more particularly, to processors and methods for power mode instruction processing, pursuant to which a generic instruction is used to control the various power saving modes in the device based on a literal value, such as a 3-bit literal associated with the power mode instruction.

Description of the Prior Art:

Processors, including microprocessors, digital signal processors and microcontrollers, operate by running software programs that are embodied in one or more series of instructions stored in a memory. The processors run the software by fetching instructions from the series of instructions, decoding the instructions and executing them. The processors run the software by fetching instructions from the series of instructions, decoding the instructions and executing the instructions. Processors, including digital signal processors, are conventionally adept at processing instructions that perform power saving. In general, power saving is achieved through specific instructions, which perform one function and one function only. Typically, the power save instruction performs a single function. For example, you may have a wait instruction, which will turn off the clocks to the CPU core, but keep the clocks to the peripherals going and another instruction called

sleep, which would perhaps turn the clocks off to everything. These power save instructions make inefficient use of processor resources and do not provide flexibility to perform power save operations.

There is a need for a new method of implementing power mode instructions within a processor that can control the sleep mode entered efficiently. There is a further need for a new method of implementing a power mode instruction that initiate a switch to the clock source configuration based on a literal value, such as a 3-bit literal, associated with the power mode instruction. There is also a need turn off power to peripheral modules on an individual basis prior to entering a sleep mode. There is a need for a processor that performs power mode instruction processing, pursuant to which a generic instruction is used to control the various power saving modes in the device based on a literal value, such as a 3-bit literal, associated with the power mode instruction.

SUMMARY OF THE INVENTION

According to embodiments of the present invention, a method and a processor for processing a power mode instruction are provided. The power mode instruction itself includes up to five different sleep modes and one run mode, each for initiating a clock source change or inhibit. This instruction may be executed in one processor cycle and with one power mode instruction employing clock transition logic within the processor to initiate a switch to the clock source configuration specified by a literal, such as a 3-bit literal.

A method of processing power mode instruction according to an embodiment of the present invention includes fetching and decoding a power mode instruction. The power mode instruction may be executed on a combination of a CPU clock source and a peripheral clock source. The instruction initiates a switch to the clock source configuration that matches a literal, such as a 3-bit literal.

In an embodiment of the present invention, the power mode instruction may be executed as a first power mode instruction. The first power mode instruction derives the CPU clock source and the peripheral clock source from a primary oscillator. Alternatively, the power mode instruction may be a second power mode instruction. The second power mode instruction derives the CPU clock source and the peripheral clock source from a low power internal RC. Alternatively, the power mode instruction may be a third power mode instruction. The third power mode instruction derives the CPU clock source and the peripheral clock source from a secondary oscillator. Alternatively, the power mode instruction may be a fourth power mode instruction. The fourth power mode instruction disables the CPU clock source and the peripheral clock source. Alternatively, the power mode instruction may be a fifth power mode instruction. The fifth power mode instruction disables the CPU clock source and derives the peripheral clock source from a low power internal RC. Alternatively, the power mode instruction may be a sixth power mode instruction. The sixth power mode instruction disables the CPU clock source and derives the peripheral clock source from a primary oscillator.

In an embodiment of the present invention, the method further includes detecting that an interrupt condition has occurred and loading the first instruction of an interrupt service routine into an instruction register for execution. Alternatively, the method further includes detecting that an interrupt condition has occurred and loading an instruction immediately following the power mode instruction into an instruction register for execution, wherein the power mode instruction disables the CPU clock source.

According to an embodiment of the present invention, a processor for performing power mode instruction processing includes a clock transition logic for initiating a switch to a clock source configuration specified by a literall and a program memory for storing instructions including a power mode instruction. The processor further includes s program counter for identifying current instructions for processing.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:
The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which embodiments of the present invention may find application;

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor, which has a microcontroller and a digital signal processing engine, within which embodiments of the present invention may find application;

Fig. 3 depicts a functional block diagram of a processor configuration for processing power mode instructions according to embodiments of the present invention;

Fig. 4 depict a method of processing power mode instructions according to embodiments of the present invention; and

Fig. 5 depicts a table of processor power modes defined by a literal according to embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

According to the present invention, a method and processor for processing a power mode instruction are provided. More particularly, a method for processing power mode instruction according to an embodiment of the present invention includes fetching and decoding a power mode instruction. The power mode instruction may be executed on a combination of a CPU clock source and a peripheral clock source. The power mode instruction itself includes up to five different sleep modes and one run mode, each for initiating a clock source change or inhibit. This instruction may be executed in one processor cycle and with one power mode instruction employing clock transition logic within the processor to initiate a switch to the clock source configuration specified by a literal.

In order to describe embodiments of power mode instruction processing, an overview of pertinent processor elements is first presented with reference to Figs. 1 and 2. The power mode instruction and instruction processing is then described more fully with reference to Figs. 3-5.

Overview of Processor Elements

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which the present invention may find application. Referring to Fig. 1, a processor 100 is coupled to external devices/systems 140. The processor 100 may be any type of processor including, for example, a digital signal processor (DSP), a microprocessor, a microcontroller or combinations thereof. The external devices 140 may be any type of systems or devices including input/output devices such as keyboards, displays, speakers, microphones, memory, or other systems which may or may not include processors. Moreover, the processor 100 and the external devices 140 may together comprise a stand alone system.

The processor 100 includes a program memory 105, an instruction fetch/decode unit 110, instruction execution units 115, data memory and registers 120, peripherals 125, data I/O 130, and a program counter and loop control unit 135. The bus 150, which may include one or more common buses, communicates data between the units as shown.

The program memory 105 stores software embodied in program instructions for execution by the processor 100. The program memory 105 may comprise any type of

nonvolatile memory such as a read only memory (ROM), a programmable read only memory (PROM), an electrically programmable or an electrically programmable and erasable read only memory (EPROM or EEPROM) or flash memory. In addition, the program memory 105 may be supplemented with external nonvolatile memory 145 as shown to increase the complexity of software available to the processor 100. Alternatively, the program memory may be volatile memory which receives program instructions from, for example, an external non-volatile memory 145. When the program memory 105 is nonvolatile memory, the program memory may be programmed at the time of manufacturing the processor 100 or prior to or during implementation of the processor 100 within a system. In the latter scenario, the processor 100 may be programmed through a process called in-line serial programming.

The instruction fetch/decode unit 110 is coupled to the program memory 105, the instruction execution units 115 and the data memory 120. Coupled to the program memory 105 and the bus 150 is the program counter and loop control unit 135. The instruction fetch/decode unit 110 fetches the instructions from the program memory 105 specified by the address value contained in the program counter 135. The instruction fetch/decode unit 110 then decodes the fetched instructions and sends the decoded instructions to the appropriate execution unit 115. The instruction fetch/decode unit 110 may also send operand information including addresses of data to the data memory 120 and to functional elements that access the registers.

The program counter and loop control unit 135 includes a program counter register (not shown) which stores an address of the next instruction to be fetched. During normal instruction processing, the program counter register may be incremented to cause sequential instructions to be fetched. Alternatively, the program counter value may be altered by loading a new value into it via the bus 150. The new value may be derived based on decoding and executing a flow control instruction such as, for example, a branch instruction. In addition, the loop control portion of the program counter and loop control unit 135 may be used to provide repeat instruction processing and repeat loop control as further described below.

The instruction execution units 115 receive the decoded instructions from the instruction fetch/decode unit 110 and thereafter execute the decoded instructions. As part of this process, the execution units may retrieve one or two operands via the bus 150 and store the result into a register or memory location within the data memory 120. The execution units may include an arithmetic logic unit (ALU) such as those typically found in a microcontroller. The execution units may also include a digital signal processing engine, a floating point processor, an integer processor or any other convenient execution unit. A preferred embodiment of the execution units and their interaction with the bus 150, which may include one or more buses, is presented in more detail below with reference to Fig. 2.

The data memory and registers 120 are volatile memory and are used to store data used and generated by the execution units. The data memory 120 and program memory

105 are preferably separate memories for storing data and program instructions respectively. This format is a known generally as a Harvard architecture. It is noted, however, that according to the present invention, the architecture may be a Von-Neuman architecture or a modified Harvard architecture which permits the use of some program space for data space. A dotted line is shown, for example, connecting the program memory 105 to the bus 150. This path may include logic for aligning data reads from program space such as, for example, during table reads from program space to data memory 120.

Referring again to Fig. 1, a plurality of peripherals 125 on the processor may be coupled to the bus 150. The peripherals may include, for example, analog to digital converters, timers, bus interfaces and protocols such as, for example, the controller area network (CAN) protocol or the Universal Serial Bus (USB) protocol and other peripherals. The peripherals exchange data over the bus 150 with the other units.

The data I/O unit 130 may include transceivers and other logic for interfacing with the external devices/systems 140. The data I/O unit 130 may further include functionality to permit in circuit serial programming of the Program memory through the data I/O unit 130.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor 100, such as that shown in Fig. 1, which has an integrated microcontroller arithmetic logic unit (ALU) 270 and a digital signal processing (DSP) engine 230. This configuration may be used to integrate DSP functionality to an existing microcontroller

core. Referring to Fig. 2, the data memory 120 of Fig. 1 is implemented as two separate memories: an X-memory 210 and a Y-memory 220, each being respectively addressable by an X-address generator 250 and a Y-address generator 260. The X-address generator may also permit addressing the Y-memory space thus making the data space appear like a single contiguous memory space when addressed from the X address generator. The bus 150 may be implemented as two buses, one for each of the X and Y memory, to permit simultaneous fetching of data from the X and Y memories.

The W registers 240 are general purpose address and/or data registers. In order to preserve data information contained in W registers 240, while processing affecting flow control is performed, the data information contained in the W registers 240 may be saved to an array of shadow registers 280. After the processing affecting flow control is performed, the data information contained in the shadow register 280 may be restored to the primary registers, thus permitting processing using the data information to resume. The W registers 240 are communicatively coupled to shadow registers 280, where each bit in the array of primary registers 240 is in communication with a bit in the array of shadow registers 280.

The DSP engine 230 is coupled to both the X and Y memory buses and to the W registers 240. The DSP engine 230 may simultaneously fetch data from each the X and Y memory, execute instructions which operate on the simultaneously fetched data and write the result to an accumulator (not shown) and write a prior result to X or Y memory or to the W registers 240 within a single processor cycle.

In one embodiment, the ALU 270 may be coupled only to the X memory bus and may only fetch data from the X bus. However, the X and Y memories 210 and 220 may be addressed as a single memory space by the X address generator in order to make the data memory segregation transparent to the ALU 270. The memory locations within the X and Y memories may be addressed by values stored in the W registers 240.

Any processor clocking scheme may be implemented for fetching and executing instructions. A specific example follows, however, to illustrate an embodiment of the present invention. Each instruction cycle is comprised of four Q clock cycles Q1 – Q4. The four phase Q cycles provide timing signals to coordinate the decode, read, process data and write data portions of each instruction cycle.

According to one embodiment of the processor 100, the processor 100 concurrently performs two operations – it fetches the next instruction and executes the present instruction. Accordingly, the two processes occur simultaneously. The following sequence of events may comprise, for example, the fetch instruction cycle:

Q1: Fetch Instruction
Q2: Fetch Instruction
Q3: Fetch Instruction
Q4: Latch Instruction into prefetch register, Increment PC

The following sequence of events may comprise, for example, the execute instruction cycle for a single operand instruction:

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: fetch operand
Q3: execute function specified by instruction and calculate destination address for data

Q4: write result to destination

The following sequence of events may comprise, for example, the execute instruction cycle for a dual operand instruction using a data pre-fetch mechanism. These instructions pre-fetch the dual operands simultaneously from the X and Y data memories and store them into registers specified in the instruction. They simultaneously allow instruction execution on the operands fetched during the previous cycle.

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: pre-fetch operands into specified registers, execute operation in instruction
Q3: execute operation in instruction, calculate destination address for data
Q4: complete execution, write result to destination

Power Mode Instruction Processing

Fig. 3 depicts a functional block diagram of a processor configuration for processing power mode instructions according to embodiments of the present invention.

Referring to Fig. 3, the processor includes a program memory 300 for storing instructions, such as a power mode. The processor also includes a program counter 305 that stores a pointer to the next program instruction to be fetched. The processor further includes an instruction register 315 for storing an instruction for execution that has been fetched from the program memory 300. The processor also includes an instruction decoder 320 and clock transition logic 325. The clock transition logic 325 switches to the clock source configuration that matches a literal, such as 3-bit literal.

The instruction decoder 320 decodes instructions that are stored in the instruction register 315. Based on the bits in the instruction, the instruction decoder 320 selectively activates clock transition logic 325 for performing the specified operation on a combination of a CPU clock source and a peripheral clock source. The clock sources include a primary oscillator 330, a secondary oscillator 335, a low powered internal RC 350. Each of the clock sources may supply power to the CPU and peripherals in accordance with a 3-bit leteral specified by the instruction. In this regard, when a power mode instruction, in accordance with a 3-bit literal depicted in Fig. 5, is presented to the instruction decoder 320, the instruction decoder generates control signals which cause the clock transition logic 325 to configure the clock sources to match the mode represented by the literals, such as a 3-bit literal.

The clock transition logic 325 may include a register, which includes a plurality of bits and a clock transition bit. The register can be written to manually, such as by a user, with data bits or can fetch an operand that will cause the clock transition logic to switch from one clock source to another clock source. The actual clock source options can be controlled employing the processor by writing to the register. Execution of a power mode instruction fetches the operand and load the operand into the transition logic 325. Writing the bits and the operand before you execute the Power Mode instruction represents the exit state from the instruction.

At any time during the processing of an instruction by the processor, an interrupt condition may occur and be serviced as determined by interrupt logic 365. When an

interrupt is serviced, program counter 350 loads into the program counter 350 the address of interrupt service routine instructions for the interrupt condition.

Fig. 4A depicts a method of processing a power mode instruction according to embodiments of the present invention. Referring to Fig. 4A, in step 400A, the processor fetches a power mode instruction from the program memory 300. Then in step 410A, the instruction decoder 320 decodes the instruction. In step 420A, the processor causes control signals to be sent to the clock transition logic 325. In step 430A, the processor executes the power mode instruction to switch the clock sources to the appropriate configuration in accordance with the decoded power mode instruction.

Fig. 4B depicts a method of processing a power mode instruction according to embodiments of the present invention. Referring to Fig. 4B, in step 400B, the processor fetches a power mode instruction from the program memory 300. Then in step 410B, the instruction decoder 320 decodes the instruction. In step 420B, the processor causes control signals to be sent to the clock transition logic 325. In step 430B, the processor executes the power mode instruction to switch the clock sources to the appropriate configuration in accordance with the decoded power mode instruction. in step 440B, an interrupt condition is detected. In step 450, the processor determines whether the interrupt is a level 7 interrupt and whether the GIE bit is set, such as with a value of 1. If the GIE bit is set and the interrupt is a level 7 interrupt, the method proceed to step 460 where the interrupt is serviced and the first instruction of the ISR is fetched. If either of the conditions are not

met, then the process goes to step 470, where the instruction immediately following the power mode instruction is fetched.

While specific embodiments of the present invention have been illustrated and described, it will be understood by those having ordinary skill in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention.